

# BASIC ADMINISTRATION TASKS

**After reading this chapter and completing the exercises, you will be able to:**

- ♦ Create and manage Linux user accounts
- ♦ Install and maintain diverse types of Linux file systems
- ♦ Manage processes on Linux using basic commands

In the previous chapter you learned what it means to be a system administrator. You also learned the role ethics and nontechnical skills play in a system administrator's daily duties. In addition, you learned about some basic concepts related to Linux system administration and became familiar with some of the most popular administrative utilities.

In this chapter you look at basic administration tasks, such as working with users, processes, and file systems. You will also learn about the utilities used to manage users, processes, and file systems in Linux.

---

## ADMINISTERING USER ACCOUNTS

In order to complete any operation in Linux, the user must first log in to a valid user account. The task of setting up and maintaining these user accounts is a large part of the work of a system administrator. In Chapter 4 you learned how to manage the initialization files for a user account. This section provides more details on how to configure and manage user accounts. In general, the more user accounts you have on your Linux system, the more work is required to keep them all running smoothly. More users also means more security risks—thus proper management and tracking of user accounts is crucial to keeping the system running securely and efficiently.

Before you can thoroughly understand the nature of user accounts, you need to understand the situations in which user accounts are *not* used. As a rule, a user account is not required when accessing a network service provided on the Linux server. For example, when a person connects to a Linux system using a Web browser, the remote Web browser does not have (or need) a user account. In fact, the remote Web browser never actually logs in to the Linux system. Instead, the Web server daemon watches for incoming requests and responds over the network without allowing the browser to have full access to a Linux user account. The Web server runs as a certain Linux user (usually as user `nobody`, to increase security)

and uses the access privileges of this user account to read files that are passed back over the network to the Web browser. Some types of network services (like telnet or FTP) require an account on the Linux system to which a client wishes to connect. Other services, such as those that send e-mail messages or request Web documents, do not.

## Types of User Accounts

You are probably familiar with the process of logging in to Linux with a user account that looks like your own name. It's important to keep in mind, however, that Linux has many user accounts with strange names that serve special purposes on the system. All of these user accounts are part of the same "system," but they have different characteristics according to how they are used by the various Linux programs. Three types of user accounts are described in the following sections.

### The root Account

As you know, the administrative account (the **superuser**) on a Linux system is named `root`. The `root` user account is created when you install Linux, at which time you normally assign a password to that account as well. The `root` user has authority to complete any operation on the Linux system, including changing any configuration information or deleting the entire operating system with a single command. The `root` user on Linux is similar to the `admin` user on a NetWare server. The administrative user account on Windows NT does not have access to all system files and resources; Windows NT does not have a user account that is truly equivalent to the Linux `root` user.

Because of the power of the `root` user, you must not log in as `root` for your normal work. Even though you are the system administrator, `root` is not intended to be your main account. Always create a separate account (normally based on your name) and log in using this account for normal work. When you need to do administrative tasks that require superuser privileges (such as creating new user accounts), you need to temporarily change to `root` account privileges, complete the administration task, and then return to your normal user account. You can temporarily change to `root` account privileges using the `su` utility.

The `su` utility (for substitute user) changes any user account's permissions to the permissions assigned to another user account. This is like logging in as a different user. If you simply type `su`, without any parameters, you change to the `root` account. If you type `su` followed by a username, you change to that user's account. This utility is useful when you need to temporarily assume the privileges of another user account for administrative purposes. For example, to assume the permissions of a user named `lizw`, you would type:

```
su - lizw
```

This command places you in the home directory of user `lizw`, with all environment settings as they would be for that user. If you omit the hyphen, you are not placed in a new directory with new environment settings. Because the `root` user has all power over the system, no password is required to use the `su` utility when logged in as `root`. Regular users must supply a password when using the `su` utility.



You must be especially careful with the root password. If an intruder obtains this password, he or she could inflict severe damage to your Linux system, including creating many security holes that will allow the intruder access to the system even if the root password is changed later.

## Regular Users

A regular user account is intended for a person who needs to log in and use the Linux system. Although a regular account can be associated with a role in an organization (you could name a user account “manager” or “designer”), user accounts are commonly associated with individuals. The name of the account reflects the name of the individual. For example, Chris Lee might have any of the following as a user account name:

- `chris`
- `lee`
- `clee`
- `chrisl`
- `cl`

User account names should be no more than eight characters. The example user account names for Chris Lee, in the preceding list, are not predefined, but depend on how you decide to set up your user accounts. It's common practice to define a standard method of converting real names to usernames. For example, an administrator might decide to combine the user's first name and last initial to create the person's username. In another scheme, the username might consist of the first initials of a user's first and last names. Some duplicate usernames may require variation from the standards you define.

## Non-Regular Users

In addition to the `root` user and the regular user accounts, Linux includes several default user account names that might appear strange to Linux newcomers. These user accounts are employed only by Linux programs and are referred to as special, or non-regular, accounts. By using a special user account, programs can better control file permissions and therefore ensure the security of the system. Most non-regular user accounts are created during the installation of Linux; others may be created by programs that you install. The non-regular user accounts created during installation of your Linux system vary depending on the services you have installed. For example, if you have installed the PostgreSQL database package, your system contains a `postgres` user; otherwise your system will not include this user. The

special user accounts you are likely to see on your system are shown in Table 8-1. Although these user accounts allow programs to control system access in their respective areas, these accounts do not have passwords or default shells defined. This means that a person cannot log in using these accounts.

Table 8-1 Non-Regular User Accounts Created by Default on Most Linux Systems

User account	Description
bin	Can be used by any program
daemon	Used by daemons
adm	Used for administrative purposes
lp	Used by the printer control daemon
sync	Used to synchronize disk updates
shutdown	Used during system shutdown
halt	Used when the system is being halted
mail	Used by the e-mail server
uucp	Used by programs related to the UUCP protocol
news	Used by the newsgroup server
operator	Can be used for system administration work
ftp	Used for anonymous FTP access
nobody	Used as a restricted access account
games	Used by game programs to control system access

## Linux Groups

Like most other operating systems, Linux allows the administrator to organize user accounts into groups. A **group** is a collection of user accounts that can be granted access to the system collectively. Assigning users to groups makes it easier to give each user access to areas of the system that match his or her specific work requirements. Permissions to use directories and files on Linux are granted to the owner of a file or directory or to the group assigned to a file or directory. Each user in Linux is assigned to a primary group. Information regarding a user's group assignment is stored with the user's account information. Users can also be assigned as members of additional, secondary groups. Information regarding a user's secondary groups is stored in the group configuration file (described in the next section). Groups can be assigned permissions, just like individual user accounts.

Many Linux systems employ User Private Groups to increase security on the system. A **User Private Group** system creates a group with a single member for each new user account that is created. The new user is the only member of the group. When a user creates a file or directory, that user's private group is assigned as the group for that file or directory; thus no other users have access to the file or directory by virtue of belonging to the same group as the user that created it. This prevents inadvertent security mishaps from making a user's files accessible to others that are part of the group assigned to a file that the user created.

To understand the nature of groups, suppose you have created a new user account called `chrislee`. Because your system employs User Private Groups, the primary group for this user is the group named `chrislee`. User `chrislee` is also assigned to the following groups: `projectleads`, `salesteam`, and `hrcommittee`. Now suppose you want to give all members of the sales team access to a particular directory or group of files. Rather than having to assign permissions to each user account individually, you can simply assign the necessary permission to the `salesteam` group. In the process, user `chrislee` will also be granted permission. (This example is illustrated in Figure 8-1.)

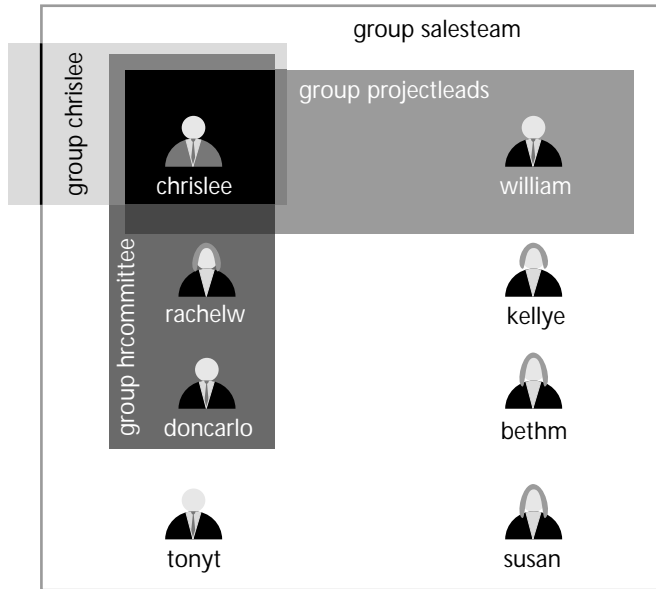


Figure 8-1 Example of groups and users

## User and Group Files

User account information is stored in the file `/etc/passwd`. In earlier releases of Linux, password information for each user was also stored in this file, hence the file's name. Because of security problems in the past, this is no longer the case. Other basic information about each user is contained in the file, however. A sample `/etc/passwd` file from a new Linux installation is shown below, followed by a description of each of the file's colon-separated fields.



The exact list of users created on a new Linux system depends on which version of Linux you are using and which features you have selected to install or activate.

```

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:
operator:x:11:0:operator:/root:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:
gdm:x:42:42::/home/gdm:/bin/bash
xfs:x:100:233:X Font Server:/etc/X11/fs:/bin/false
nwalls:x:500:500:Nicholas Wells:/home/nwalls:/bin/bash

```

The following list describes the fields in the list above. The last line of the file (the user `nwalls`) is used as an example.

- User account name (`nwalls`): the name used by a person to log in to Linux.
- Password (`x`): the password for each user was formerly stored in this field in encrypted form. An `x` in this field indicates that the shadow password system is in use, in which case the password information is stored in the file `/etc/shadow`. You will learn more about shadow passwords later in this chapter.
- User ID number, or UID (the first 500): a number from 0 to 65,535 that uniquely identifies this user on this Linux system. The number is arbitrary and normally is automatically assigned by the utility used to create a new user account.
- Group ID number, or GID (the second 500): a number from 0 to 65,535 that uniquely identifies the primary group for this user account. The GID must correspond to a group defined in the `/etc/group` file (described below).
- The user's real name (`Nicholas Wells`): a complete name (or a comment for non-regular users). Spaces are permitted in this field. If the user account was created for a certain role in the organization, other text can be placed here instead, such as "Database Administrator."
- Home directory (`/home/nwalls`): the position in the Linux file system that will be used as the current working directory when the user first logs in.
- Default shell (`/bin/bash`): the program that runs automatically when the user logs in. The default setting for this field is `/bin/bash`, which runs the `bash` shell. If a user prefers a different shell (such as the **Korn shell** or **C shell**), this field can be changed to accommodate that. This field can also be used to start a nonshell program to restrict the user's actions in the system.

Although you can edit the `/etc/passwd` file directly in a text editor, this is not a good idea. Advanced security measures that have been added to your distribution of Linux may make any alterations to the `passwd` file invalid. In addition, there is a small risk that another program might be trying to edit user information at the same time and create a conflict. Instead of a text editor, use the programs described in the following sections to update the user account file. If you need to use a text editor to correct a problem in the file, try the special editing program `vipw`. (This program is basically a copy of the `vi` editor that automatically loads the `passwd` file.)

Groups on a Linux system are defined in the `/etc/group` file. A sample of this file is shown here, with the fields in the file (again separated by colons on each line) described in the following list.

```
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
tty::5:
disk::6:root
lp::7:daemon,lp
mem::8:
kmem::9:
wheel::10:root
mail::12:mail
news::13:news
uucp::14:uucp
man::15:
games::20:
gopher::30:
dip::40:
ftp::50:
nobody::99:
users::100:
floppy:x:19:
console:x:101:
gdm:x:42:
utmp:x:102:
pppusers:x:230:
popusers:x:231:
slipusers:x:232:
slocate:x:21:
xfs:x:233:
nwells:x:500:
rsolomon:x:501:
authors:x:502:rsolomon,nwells,jsmith
```

- The name of the group: this field cannot contain spaces. Avoid names more than eight letters long.
- Group password: this field is either blank or `x` (meaning the password is stored in another location). Group passwords are rarely used.
- Group ID (GID) number: this number uniquely identifies this group within the Linux system. Group numbers are automatically assigned when you create a new group, though you can specify a number if you prefer.
- Members of the group: This field identifies members of the group. Note in the sample file that many groups do not have member users defined. A program may be able to assume the permissions of the group using system calls (programming instructions), but no user is part of the group by virtue of logging in. Some of the groups (such as `sys` and `adm`) have a comma-separated list of users as the last field. In addition to the two User Private Group items (for `nwells` and `rsolomon` in this sample file), a standard group named `authors` has been added to this default installation.



Some UNIX and Linux systems employ a special group called **wheel**, which has special administrative powers; it is essentially a reduced version of the `root` account. On some systems a user must be a member of the `wheel` group in order to use the `su` command to change to the `root` account permissions. Although Linux includes a `wheel` group by default (with `root` as the only member), no special features or privileges apply to the `wheel` group in Linux.

## Shadow Passwords

All programs and users may need to access the list of users on the system stored in the `/etc/passwd` file. However, if the encrypted password text is readable by many users, it may be subject to attack, allowing unauthorized use of someone's account.

To counteract this problem, the passwords for Linux user accounts are no longer stored in the `/etc/passwd` file. Instead, they are commonly stored in a file called `/etc/shadow`. Systems that make use of this file are said to be using the **Shadow Password system**. This file can only be read by the `root` user (and special programs such as the `login` routine). This tighter security protects all user passwords from the open access formerly allowed with the `/etc/passwd` file.

A sample `/etc/shadow` file is shown below. Fields on each line are separated by colons, as in the `/etc/passwd` file. The first field is a user account name that must correspond to a user account in `/etc/passwd`. The second field is the encrypted password text. Additional fields configure password security information for the user account on that line. For the many non-regular user accounts (such as `bin` and `daemon`), an asterisk in the second field indicates that the account has no password. No user can log in to an account with no password.

```

root:$1$xl05lRMK$oklXHuoBjHH7JmiVdk/fQ.:10815:0:99999:7:-1:
-1:134538444
bin:*:10815:0:99999:7:::
daemon:*:10815:0:99999:7:::
adm:*:10815:0:99999:7:::
lp:*:10815:0:99999:7:::
sync:*:10815:0:99999:7:::
shutdown:*:10815:0:99999:7:::
halt:*:10815:0:99999:7:::
mail:*:10815:0:99999:7:::
news:*:10815:0:99999:7:::
uucp:*:10815:0:99999:7:::
operator:*:10815:0:99999:7:::
games:*:10815:0:99999:7:::
gopher:*:10815:0:99999:7:::
ftp:*:10815:0:99999:7:::
nobody:*:10815:0:99999:7:::
gdm:!!:10815:0:99999:7:::
xfs:!!:10815:0:99999:7:::
nwell$:$1$3gWKUouQ$7XUsJWpIwtqLUoWlmVvN1:10816:0:99999:7:-1:
-1:134538436
rsolomon: 1J42Wuip3dYAh8$1pvNMAVK$UsrD6O90:10817:0:99999:7:-1:
-1:134538412

```

On some Linux systems, user password security goes even further, with various systems available that can hide and encrypt passwords. These systems are beyond the scope of this book, however.

## Creating New User Accounts

New user accounts can be created using any of several methods. The most rudimentary is to edit the `/etc/passwd` file and then use the `mkdir` command to create a new home directory. This has several disadvantages, however:

- Editing the `/etc/passwd` file can create a conflict with another program trying to edit the file, as already stated.
- Hand editing can introduce syntax errors. The new user account may not work, and the errors might make other user accounts invalid as well.
- Advanced security systems that store user information in nonstandard locations may not be affected by direct changes to the `/etc/passwd` file.
- Using one of the other methods is easier and can be included in system administration scripts.

User administration is one of the main tasks of a system administrator, so many graphical tools are available. For instance, you may choose to use one of the graphical user configuration tools

shown in Figure 8-2. You had an opportunity to use the `LinuxConf` graphical utility in Chapter 4, when you created a user account for yourself after installing Linux. These utilities all work with the same core configuration files. Some may allow group and password management in addition to user account management.

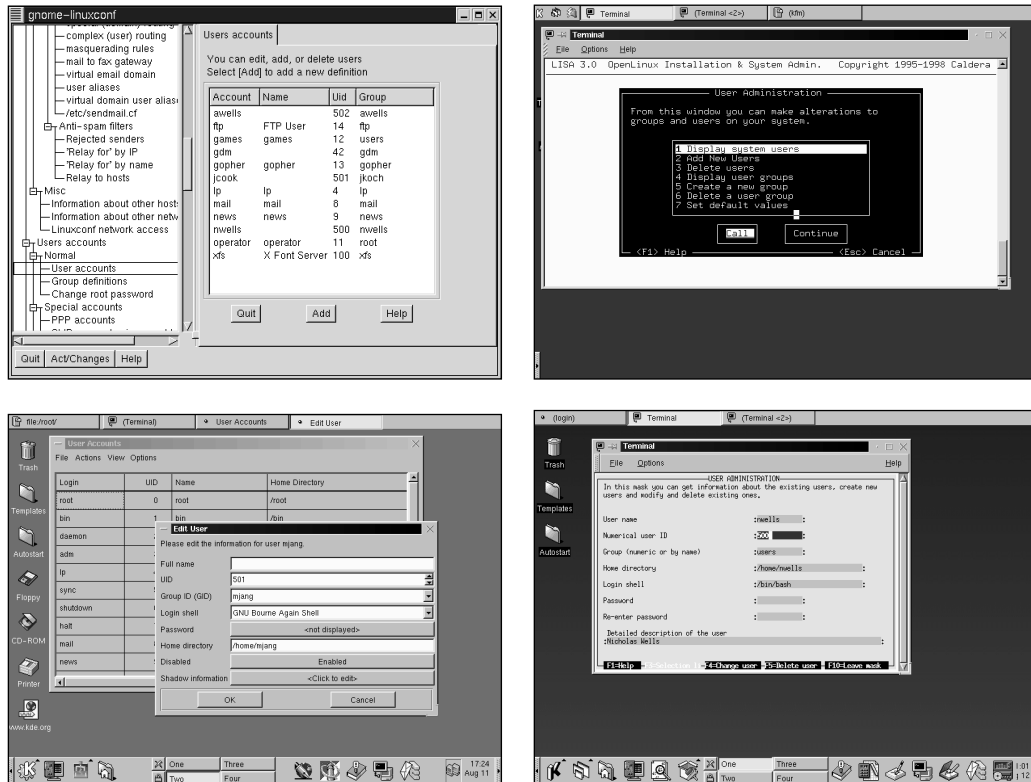


Figure 8-2 Graphical user account configuration tools

The most secure method of managing user accounts, however, is the `useradd` command. You will have a chance to practice using the `useradd` command in the hands-on projects at the end of this chapter.

The `useradd` command allows you to automate user creation, update user accounts, and take advantage of various options when creating users. If additional security is added to your Linux system, an updated version of `useradd` is normally included as the preferred method of adding new user accounts.



On some Linux systems, a script named `adduser` is available. Using a script to create new users is less secure than using the `useradd` command. On the latest Linux systems (such as Red Hat 6), the `adduser` command merely points to the `useradd` command.

To add a user with `useradd`, you must be logged in as `root`. Along with the command name, include the name of the new user account as a parameter. For example, to add a new account called `rsolomon`, you would use this command:

```
useradd rsolomon
```

The default user account settings are used to create the account and a home directory. These defaults are stored in the `/etc/login.defs` file and in the `/etc/default/useradd` file.

Specific options can be added to the `useradd` command. These options override any default settings for the user account that you are creating. For example, if you want to include the user's full name in the command field (generally a good idea), you can use the `-c` option. The `-g` option defines the primary group for the new user. A command incorporating these two options would look like this:

```
useradd -g sales -c "Raley Solomon" rsolomon
```

In this example, the value of the `-c` parameter is `Raley Solomon`. Because this value includes a space, it must be enclosed in quotation marks so the `useradd` command does not interpret the part after the space (`Solomon`) as the next command parameter.

Table 8-2 shows the options for the `useradd` command.

Table 8-2 Useradd Command Options

Option	Description	Example
<code>-c</code>	Defines a user's full name or other comment for this account	<code>useradd -c "Jose Carrera" josec</code>
<code>-d</code>	Specifies the home directory path (useful mostly for special user accounts that use a nonstandard home directory location).	<code>useradd -d /usr/home/ josec</code>
<code>-e</code>	Specifies the date this user account will expire (and be disabled automatically). Used for temporary accounts.	<code>useradd -e 03/15/01 josec</code>
<code>-f</code>	Specifies the number of days after the password expires until the account is disabled.	<code>useradd -f 7 josec</code>
<code>-g</code>	Specifies the primary group for the new user (either the group's name or its unique GID number can be used).	<code>useradd -g ops josec</code>

Table 8-2 Useradd Command Options (continued)

Option	Description	Example
-G	Adds a list of additional groups that the new user should be made a member of (this information is stored in the <code>/etc/group</code> file, not in <code>/etc/passwd</code> ).	<code>useradd -G teamlead,party,emt josec</code>
-m	Forces creation of the user's home directory, even if the default settings do not include creating a home directory.	<code>useradd -m josec</code>
-M	Does not create a home directory, even if the default is set to include one.	<code>useradd -M josec</code>
-n	Disables the User Private Group feature so that a group matching the new username is not created.	<code>useradd -n josec</code>
-s	Sets the user's login shell. The default shell in Linux is <code>bash</code> . The complete path to another shell program can be used with this option.	<code>useradd -s /bin/zsh josec</code>
-u	Sets a specific numeric value for the user ID of the new user. (Normally a UID is selected automatically—use this option if you need to force the use of a specific UID number.)	<code>useradd -u 509 josec</code>

To display the default settings for the `useradd` command, use the `-D` option. Typical output of the `-D` option is shown here:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

The information returned by the `-D` option is described in the following list:

- **GROUP:** the group ID number for the group that all new users will be placed in (as a primary group) if no other is indicated when the user is created.
- **HOME:** the path in which home directories for new users will be created.

- **INACTIVE**: the number of days after the password for the new account expires that the account will be disabled. Using a value of -1 for this field disables this option (the user account will *not* be disabled).
- **EXPIRE**: the expiration date for a new user account.
- **SHELL**: the path and program name for the default shell (command-line interpreter) to be used by a new user account.
- **SKEL**: the path to the skeleton directory used to fill a new home directory with basic files (this directory is discussed later in this section).

You can also use the `-D` option to update the defaults that will be used in the future for all new user accounts. For example, to change the default shell so that all new users will use the C shell instead of the `bash` shell, use this command:

```
# useradd -D -s /bin/csh
```



If you prefer, you can edit the `/etc/default/useradd` file directly in a text editor.

The items you are most likely to need to update in the default user creation settings are the login preferences—that is, the settings for how long a password can be used, how many days after expiration before the account is disabled, and so forth. Because all of these options are part of Linux security, they are not described in detail here. They are documented in the `/etc/login.defs` file and in the man page for `useradd`.

After adding a new user account, check that the user's home directory has been created. For example, after creating a user with the command `useradd jsmith`, you should check that the directory `/home/jsmith` exists. Depending on how you have set up e-mail accounts on your network, each user may also need a file to hold incoming e-mail in the `/var/spool/mail/` directory. For example, after creating a new account with the command `useradd jsmith`, you may need to also create the mail file for this user with the `touch` command (which creates an empty file or updates the access time of a file). In this example, the `touch` command would be: `touch /var/spool/mail/jsmith`.

## Changing User Passwords

Before anyone can log in using a new account, the account must be assigned a password. After the password is assigned, the account is ready for normal use. A password is not defined by `useradd` when a new user account is created.



Some of the graphical tools shown in Figure 8-2 can be used to create a password for the new user account. It is not possible to create passwords with the command-line utility `useradd`.

The `passwd` command is used to change (or initially set up) a password on a user account. (This command has the same name as the `/etc/passwd` file.) To use this command as `root`, include the name of the user account whose password you need to configure. You must then enter the new password twice to be certain you have not made a typing error.

Suppose you have already created a new user named `lizw`, and you want to set this user's initial password or change this user's password. For either task, you must do the following:

1. Make sure you're logged in as `root`.
2. Enter the command `passwd lizw`. The following text then appears on the screen:

```
Changing password for lizw
New UNIX password:
```

(Note that the word *UNIX* indicates the type of password system Linux is using.)

3. Type the new password for the Linux user account and press **Enter**. Nothing appears on screen as you type, so work carefully. The following text appears when you press Enter:

```
Retype new UNIX password:
```

4. Type the new password a second time, exactly as you typed it the first time. This verifies that the password was entered as you intended to type it, without any typing mistakes. When you press Enter the second time, the following text appears:

```
passwd: all authentication tokens updated successfully
```

If you enter a password that is a poor choice (such as *password*, the username, or a simple word from the dictionary), you see a message stating `BAD PASSWORD`. Although this message should cause you to reconsider the password, the password is still changed. For a temporary password on new accounts, almost anything will do. Popular choices include the user's account name (`lizw` in this example), the word *password*, *change.me*, or something similar.

The standard procedure is for a system administrator to assign an initial password to a new account using the steps just given, thus enabling the user account for regular use. The administrator should communicate the password to the new user, who should then immediately select a new password that is unknown to the `root` user or to any other users.

The user can change his or her password by using the `passwd` command without any parameters. Thus, after `lizw` has logged in, she can change her own password as follows:

1. Type the command `passwd`. The following text then appears on the screen:

```
Changing password for lizw
(current) UNIX password:
```

(Note that the word *UNIX* indicates the type of password system Linux is using.)

2. Type the current password for the `lizw` account and press **Enter**. Nothing appears on screen as you type, so work carefully. The following text appears when you press Enter:

```
New UNIX password:
```

3. Enter a new UNIX password, typing carefully (nothing appears on screen as you type). When you press Enter, the following text appears:

```
Retype new UNIX password:
```

4. Type the new password a second time, exactly as you typed it the first time. This verifies that the password was entered as you intended to type it. When you press Enter, the following text appears:

```
passwd: all authentication tokens updated successfully
```

Although the root user can use any word as a password, the default settings do not allow regular users to change their passwords to something like *password* or their user account name. A message stating BAD PASSWORD will appear if a poor password choice is entered, and the password will not be updated. The root user can change the root password by using the passwd command without including a user account name.

As the system administrator, you must explain to users the importance of changing their passwords immediately after a new account is created for them. Passwords should be changed monthly, even if the Linux system does not enforce frequent changes. This lessens the danger that someone may discover a user's password and be able to continue using it. Good passwords have these characteristics:

- They are at least 5 characters long, though a 7- to 10-character password is *much* more secure.
- They include digits or punctuation marks. A common trick is to substitute the number 1 for the letter *l* and the number 0 for the letter *o*, but this is too well known to add much security.
- They mix upper- and lowercase letters in nonstandard ways.
- They are easy for the account owner to remember, but hard for anyone else to guess—even someone who knows the account owner well.
- They are not created from a simple manipulation of a word found in a dictionary or the name of a person or place.

A password that is hard to remember is probably hard for someone else to discover, but it doesn't help security much if the password is written on a note taped to the computer monitor. Creating a password that is pronounceable (with punctuation added in the middle) will help you to remember it. Three good examples of passwords are:

- miCru%norMouse@
- BLAST-!t-ALL
- call=9LL&nOw

As you choose a password for your account, especially the root account on the Linux system, remember that you will be dealing with many different passwords. These include the root password, a personal account password, plus passwords for other parts of your life, such as bank accounts, Web pages, and voice mail codes. If these passwords and codes are identical or even similar, discovery of one of your passwords could jeopardize the security of many different areas.

## Creating New Groups

Although modifying the `/etc/group` file in a text editor does not pose as great a danger as editing `/etc/passwd`, the preferred method for adding a new group is to use the **groupadd** command. This command is used much like the **useradd** command, but it supports fewer options.



Graphical tools designed for creating users often allow you to create groups as well. These tools are useful if you prefer to work in a graphical environment. However, you should learn about the **groupadd** command as a backup and for troubleshooting, because the graphical tools rarely allow you to do much besides simply creating and deleting groups.

To add a new group, include the group name as a parameter, as follows:

```
groupadd managers
```

If you need to use a specific GID number for the new group, you can include it with the `-g` option. For example:

```
groupadd -g 919 managers
```

## Modifying User Accounts

After setting up user and group accounts as described in the preceding sections, you will find occasions when you need to modify or update the account information. To do this, use the **usermod** (for *user modify*) command or **groupmod** (for *group modify*) command. The **usermod** command uses the same options as the **useradd** command, but it operates on an existing user account. To use the **usermod** command to update a user's account information, type **usermod** followed by one of the **usermod** parameters and a value for that parameter. For example, suppose **lizw** gets married and wishes to have her full name changed from Liz Wells to Liz Osowski on her employment records and user account. Using the `-c` option, as with the **useradd** command to change the Comment field of the user account, the command to update the **lizw** account to include the new name would be:

```
usermod -c "Liz Osowski" lizw
```

You can change the user's login name from **lizw** to **lizo** with the `-l` option:

```
usermod -l lizo -d /home/lizo lizw
```

Using the `-l` option alone leaves the home directory as it was before (`/home/lizw`). By using the `-d` option shown above, the home directory path for the user account is updated as well (to `/home/lizo`). Note that the **usermod** command cannot be used to change the directory name. After using the **usermod -d** command, you must change the actual directory name as follows:

```
mv /home/lizw /home/lizo
```

As another example, suppose you created an account for a new employee, using the default settings, and then discovered that the new employee prefers to use a different login shell and

needs to be part of several additional groups to accommodate her job responsibilities. The command to update the user account would be something like this:

```
usermod -G taskforce,marketing -s /bin/tcsh srubenst
```

## Automating Home Directory Creation

When you create a new user account, it's very useful to include basic configuration files in the new user's home directory. This information might include:

- Company document templates and calendars
- Environment settings to access department printers and servers
- Terminal settings to make Linux work well with desktop PCs
- Commands (scripts) to automate basic tasks and set up the user's system each time the user logs in

By using the `/etc/skel` directory, you can automatically copy files and thus apply settings to each new user account as you create the account. When you use `useradd` (or most graphical user creation utilities), all of the files in the `/etc/skel` directory are copied into a new user's home directory when the account is first created. As the system administrator, you should place files in `/etc/skel` when you first install Linux so that those files are automatically placed in each user's home directory that you create later with the `useradd` command. Because these are likely to be configuration files, many of them are hidden. By using the `ls -la` command, you can list the contents of the `/etc/skel` directory. Here is one version:

```
$ ls -la /etc/skel
total 12
drwxr-xr-x  4 root  root  1024 Jun 10  08:12 .
drwxr-xr-x 31 root  root  3072 Aug  9  14:13 ..
-rw-r--r--  1 root  root  1422 Mar 29  09:08 .Xdefaults
-rw-r--r--  1 root  root    24 Jul 13  1994 .bash_logout
-rw-r--r--  1 root  root   230 Aug 22  1998 .bash_profile
-rw-r--r--  1 root  root   124 Aug 23  1995 .bashrc
drwxr-xr-x  3 root  root  1024 Jun 10  08:12 .kde
-rw-r--r--  1 root  root   966 Apr 16  14:45 .kderc
drwxr-xr-x  5 root  root  1024 Jun 10  08:12 Desktop
```

The files shown here are used for the X Window System (graphical interface), the `bash` default shell, and the KDE Desktop (which was installed on this system). If you want to have other files included in each user's home directory, simply copy those files to `/etc/skel`.



The files from `/etc/skel` are copied to a user's home directory when the account is created. When you add files to `/etc/skel`, they are not added automatically to the home directories of all existing user accounts; only user accounts created after the new files are added will include them. For existing accounts, you must copy any additional files to the home directories manually.

## Creating Aliases to Ease User Angst

As you add files to the `/etc/skel` directory in order to prepare an environment for new users, you may also want to edit the existing startup files to add functionality or ease the transition to Linux. One of the best ways to do this is by adding aliases to the `.bashrc` startup script. The commands in this script are executed each time a user starts a session (logs in or opens a new command-line window). By adding aliases to this file, you can define pseudo-commands that may make it easier for users to work with Linux (you may even want to add some to your own environment).

By using the `alias` command, you can give any Linux command another name—an alias. For example, you can create an alias named `copy` for the `cp` command. Then whenever you enter `copy` on a command line, the `cp` command is executed. To use the `alias` command to define another name for a command, you include the new command name (the alias) followed by an equal sign and the real command (which can include command options if you wish). The real command should be enclosed in quotation marks if it includes spaces. For example, this alias command makes the `copy` command (which doesn't really exist in Linux) execute the real command `cp`:

```
alias copy=cp
```

You can execute this `alias` command on any command line. By adding it to the `.bashrc` script, the `alias` command is executed automatically as every command-line session starts. After you enter this `alias` command, the `copy` command will always be interpreted as `cp`. How is this done? The shell (`bash`) actually substitutes the string `cp` whenever you enter the string `copy`. As with the rest of Linux, these strings are case sensitive, so entering `COPY` won't have the same effect as entering `copy`. As a rule, adding a few of these aliases can make it easier for users to become comfortable with Linux commands.



Don't confuse the text substitution aliases described here with other types of links or substitutions that may also be called aliases or links. Examples include symbolic links in the file system and e-mail aliases used by an e-mail server.

If you add aliases to the `.bashrc` file, they are only in effect if the user runs the `bash` shell. Similar startup files can be created for other shells, such as `.kshrc` for the Korn shell and `.cshrc` for the C shell. Each of these is a hidden file (and thus the filename begins with a period).

Aliases serve many purposes. In addition to making DOS commands available, as in the example for `copy`, you might add aliases to do the following:

- To shorten commonly used commands. For example, if you must regularly use the command `cd /mnt/samba/datafiles/project/`, you could create an alias that allows you to simply type `cdp`. This command sets up the alias:

```
alias cdp="cd /mnt/samba/datafiles/project/"
```

- To fix typing errors. For example, if you habitually type `sl` instead of `ls`, create an alias that changes `sl` to `ls`. This command sets up the alias:

```
alias sl=ls
```

- To make operations safer by including options to confirm file deletions. A common feature is to have an alias named `rm` that refers to `rm -i`, so that you must confirm each file that is removed. This command sets up the alias:

```
alias rm="rm -i"
```

To see a list of the aliases that are active in your environment at any time, use the `alias` command without any parameters, as follows:

```
$ alias
```

You can also review the `.bashrc` file in your own home directory or in `/etc/skel` to see the default aliases configured on your system. (On some Linux systems, aliases are configured in the systemwide files `/etc/bashrc` or `/etc/profile`.)

## Setting Up Environment Variables

In addition to aliases, many users will require environment variables that make it possible to access various programs and services. **Environment variables** are named values that any program can access. For example, when a database program is executed, it may expect an environment variable named `DB_DIR` to indicate the path where the database files are located. If that environment variable is not specified, the database will not be able to operate. In such a situation, each user's environment must include a definition of the `DB_DIR` environment variable.

Many Linux programs rely on environment variables to store configuration information. Rather than maintain a configuration file, a program's documentation may specify several environment variables that the user can set to alter how the program operates. The `bash` shell is a good example of this. The `bash` shell uses many environment variables to determine how features of the shell are used. For example, to change the prompt used at each command line, a user simply alters the environment variable `PS1`. To alter how often the shell checks for new e-mail messages, a user changes the value of the `MAILCHECK` variable. The manual page for `bash` lists over 50 environment variables that a user or program can use to learn the status of `bash` or to affect how `bash` operates.

To set an environment variable from the command line, use the `export` command with the variable name and value. For example:

```
export DB_DIR=/opt/database/
```

However, to avoid the need to enter this command each time a user logs in, this command can be added to the configuration files described previously: `.bashrc` in the home directory, or `/etc/bashrc` if the environment variable is needed by all users.

## Disabling User Accounts

At times you will need to disable a user account—either temporarily or permanently. Reasons for disabling a user account include:

- An employee has left the organization (permanent deletion of the account)
- An employee is on vacation (temporary disabling as a security precaution)

- A guest user has not paid for the account or for computer time (temporary, perhaps permanent later on)
- An employee is under disciplinary action and is not allowed to access company information (temporary, perhaps permanent later on)

To temporarily disable a user's account, you can simply change the password so that the user can no longer log in. This can be done with the `passwd` command as described previously. However, be sure to use a password that is not easy to guess, rather than a simple password like `change.me`.

If you are concerned about having an active account with only a new password as security, you can edit the `/etc/shadow` file in a text editor and place an asterisk before the encrypted password. This saves the password (because you can simply remove the asterisk later to reenable the account). But while the asterisk is part of the password, Linux will not allow anyone to log in to the user account. The line in `/etc/shadow` before the edit might look like this:

```
nwells:$12$tJhxVO2kUgVU2/o0434jj0:10799:0:99999:7:-1:-1:134538468
```

And after the edit it looks like this:

```
nwells:*$12$tJhxVO2kUgVU2/o0434jj0:10799:0:99999:7:-1:-1:134538468
```



The other fields of the `/etc/shadow` file are described in the manual page for this file but are not described here because they relate explicitly to system security.

If you decide to permanently delete a user's account, use the `userdel` command with the user account name. For example:

```
userdel lizo
```

This command removes the user named `lizo` from the user database (`/etc/passwd` or a similar secure file). As a result, the user will no longer be able to log in because the user account no longer exists. However, keep in mind that the `userdel` command does *not* remove the user's home directory or its contents. As a result, it is possible for the administrator to review or save the information contained in the home directory of a disabled user account. Be aware, however, that if an employee is leaving the organization, friends may be able to access part of the former employee's home directory (because of common group membership, for example) and pass files to that person. It's generally a very good idea to archive or otherwise remove or relocate the home directory of a deleted user account as soon as possible after deleting the account.

## MAINTAINING FILE SYSTEMS

When you install Linux, you create the **root** file system in which the operating system is stored. The term **file system** refers to an organized set of data that can be accessed via the standard Linux directory structure. The command-line instructions (such as `cd /home/nwells`) that refer to the directory paths in Linux provide access to data stored in an underlying file system located on a hard disk or other physical device.

The **root** file system within which Linux is installed is normally located on one of the computer's main hard disks. Even a basic Linux system uses many other different file systems, however. Each one provides access to a different set of information. In some operating systems, file systems are accessed using drive letters or special network access tools, but as you have seen in previous chapters, all Linux file systems are accessed as part of a single directory tree, starting with the root directory. The root directory is always indicated by a forward slash `/`.

To access a file system in Linux, it must first be mounted into the root directory structure. Even the **root** file system must be mounted—although this occurs during the initialization of Linux at boot time. Other special file systems (listed in Table 8-3) are also mounted during initialization. You can set up additional file systems to be mounted during initialization, or you can mount them manually after the system has booted. The Linux directory structure always provides access to multiple file systems; each one is accessed via a different directory path. The sample setup in Figure 8-3 shows how different parts of the directory structure can be located on different physical devices.

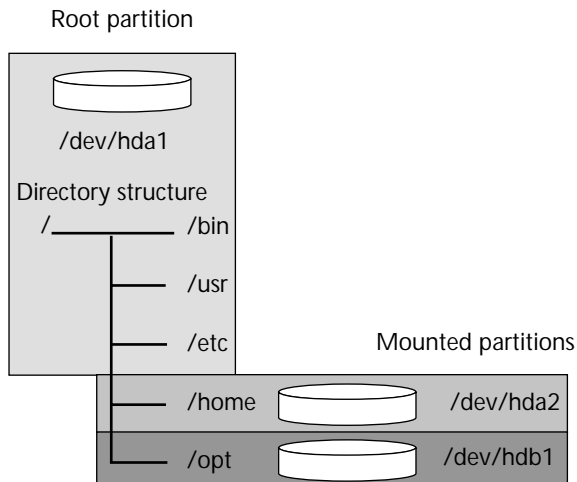


Figure 8-3 File systems mounted in the directory structure

Some of the special file systems that Linux mounts automatically after a standard installation are shown in Table 8-3. The term **mount point** refers to the path in the directory structure where the data in a file system can be accessed. The data stored within some of these file systems can be viewed using the **cat** command. This command dumps the contents of a file to STDOUT (which normally appears on the screen). For example, you can use the following command to view information within the **proc** file system:

```
cat /proc/meminfo
```

Table 8-3 Automatically Mounted File Systems

File system	Mount point	Description
swap	No mount point. This is a special file system used only by the Linux kernel.	Used to create virtual memory, allowing the Linux kernel to work as if the amount of system memory available is the sum of RAM and the <b>swap</b> file system.
proc	/proc	Provides up-to-date information about the kernel and all processes running on Linux.
auto	/auto	Automatically mounts a device when a request is made to the device. (This is called the automounting file system or the automount daemon— <b>amd</b> . It is installed automatically with most Linux systems.)
root	/	Serves as the base of a running Linux system. The <b>root</b> file system cannot be unmounted unless you first shut down Linux.

You use the **mount** utility to view all of the file systems currently available to the system. Using the **mount** command without any parameters displays a list of the currently mounted file systems, as follows:

```
$ mount
/dev/hda4 on / type ext2 (rw)
/dev/hda2 on none type swap (rw)
/proc on /proc type proc (rw)
brighton:(pid455) on /auto type auto
intr,rw,port=1023,timeo=8,retrans=110,indirect,map=/etc/amd.local
dev)
```

The output from the **mount** command includes fields, from left to right, as described in the following list:

- The device where the file system is located (such as **/dev/hda4** on the first line of the output above, which refers to a hard disk partition)
- The path in the directory structure where the file system can be accessed (such as **/** on the first line of output above)
- The type of the file system, which indicates the format of data stored on the file system (**ext2** is the type on the first line above)

- The options that apply to the file system. These are described in detail in Table 8-5. The options on the first line of output above are `rw`, indicating that the file system is mounted for both reading and writing of data.

Managing Linux file systems is critical to running a successful Linux system. Although file systems as a rule don't require much day-to-day maintenance, the more people using a Linux system and the more crucial the data stored on that system, the more important it becomes to track and maintain the file systems.

The next sections describe how to manage file systems to provide disk space and stability for all Linux users. Additional information on safeguarding file systems is provided in Chapter 9. Chapter 14 describes how to back up file system data.

## Checking File System Status

File systems that are used regularly tend to become disorganized and to fill up with data as users create new files. If the `root` file system of Linux becomes full, the Linux kernel cannot function and the system will crash. If the space where users' files are stored becomes full, users will not be able to complete their work.

By using the `df` utility at a Linux command line, you can display the file systems that are mounted in Linux and see the space used on each one. The `df` utility only displays regular file systems, not special file systems like `/proc`. The following sample output from `df` shows two file systems that are dangerously full.

```
$ df
File system    1024-blocks    Used      Available  Capacity  Mounted on
/dev/hda4      956173        895614    11160      99%       /
/dev/hda3      1018329       901074    64643      93%
/opt
sundance:/a    2017438       1210459   806979     60%       /a
```

The fields output by the `df` command, from left to right, are described in the following list:

- The device where the file system is stored. This is normally either a hard disk device name or a networked location (as in the last line of the sample output).
- The number of 1 KB blocks on the device. This indicates the file system's overall size. For example, in the sample output, the size of the three devices currently mounted are approximately 1 GB, 1 GB, and 2 GB, respectively.
- The number of 1 KB blocks that are used on the device.
- The number of 1 KB blocks that are free on the device.
- The percentage of capacity reached so far (percentage full) for the device. This is the critical number. If this value is approaching 100%, action needs to be taken.
- The location in the directory structure where this device is accessed.

If a file system is becoming full, you probably won't have the luxury of shutting down the Linux system while you figure out what to do. The busier your Linux system, the quicker a file system can fill up as multiple users download files, create new documents, and so forth.

To understand how quickly the Capacity percentage can increase, consider this example: If the hard disk partition where a file system is located is only 1 GB in size, one percent of the file system is 10 MB, not even enough for one large application to be downloaded. If you have a 100 GB file system, one percent is 1 GB, which may be enough space to continue running for a short time. Of course, if you need 100 GB of storage, you can probably use 1 GB quite rapidly. The `df` command run on a major ISP might look more like the following example. After reviewing this, you should be able to see why larger systems require more careful maintenance procedures to keep them running smoothly.

File system	1024-blocks	Used	Available	Capacity	Mounted on
/dev/dsk/c0t3d0s0	229610	110187	119423	48%	/
/dev/dsk/c0t3d0s6	306954	245175	61779	80%	/usr
/dev/dsk/c0t0d0s0	5783718	3378119	2405599	58%	/var
/dev/dsk/c0t1d0s7	2663048	1983612	679436	75%	/space1
/dev/dsk/c0t1d0s6	533992	232744	301248	44%	/usr/local
nfs.isp.com:/home	69837128	42182931	27654197	60%	/home
mail.isp.com:/var/mail	10766840	8172635	2594205	76%	/var/mail

If you see that a file system is nearing capacity, you can immediately free space by performing one of the actions in the following list. Remember, however, that you must free space in the directories where the file system is mounted. For example, if you have a separate device such as a hard disk partition mounted at the `/home` directory and the partition is almost full, you must free space in `/home` or its subdirectories. Freeing space in `/tmp` won't help.

- Look for large or numerous files in the `/tmp` directory that can be deleted.
- Look for large or numerous files in the `/var` subdirectories, especially in `tmp/` and `spool/`.
- Move the system log file (`/var/log/messages`) to another file system that isn't as full.
- See if any of the user subdirectories are using an inordinate amount of disk space.
- Consider deleting unused archive files that are backed up or even applications that you can reinstall later when space is not critically short.

While you need to be very careful as you delete files, you may have to act quickly in response to an overly full file system. In using the techniques just listed, the `du` utility can be a big help. The `du` utility provides disk usage information on a directory tree. When you run `du`

at a Linux command line, you see a summary report of the space used by that directory and each of its subdirectories. A few sample lines from the output of `du` are shown here:

```
$ du
22      ./public_html
2228    ./Public/shell_programming
2229    ./Public
2       ./Desktop/Autostart
2       ./Desktop/Trash
8       ./Desktop/Templates
13      ./Desktop
1       ./kde/share/apps/kfm/tmp
1       ./kde/share/apps/kfm/bookmarks
6       ./kde/share/apps/kfm
1       ./kde/share/apps/kppp/Rules
1       ./kde/share/apps/kppp/Log
3       ./kde/share/apps/kppp
10      ./kde/share/apps
15      ./kde/share/config
1       ./kde/share/icons/mini
2       ./kde/share/icons
1       ./kde/share/applnk
1       ./kde/share/mimelnk
30      ./kde/share
31      ./kde
1       ./archive
2337    .
```

The number at the far left indicates how many 1 KB blocks are used by the subdirectory. Every subdirectory is shown separately, with totals for the parent directory. For example, the line showing 13 KB for the `Desktop` directory includes the sum of the `/Desktop/Autostart` directory, the `/Desktop/Trash` directory, and the `/Desktop/Templates` directory, as well as any files located in the `Desktop` directory itself. So by looking at the last line (the period indicates the current directory), you can see how much space is used by the entire directory tree. More importantly, if you need to manage how space is used on a file system, you can see which subdirectories are consuming space, even if they are buried deep in the directory structure.

Suppose you wanted to see if any single home directory consumed more than 10 MB of space. The output of the `du` command is given in KB, so we calculate that 10 MB is equal to roughly 10,000 KB of space. The following commands would change you to the home directories (on most Linux systems—the location is configurable) and list any large directories for you:

```
$ cd /home
$ du | grep ^.....[0-9]
```

If any line of the output of `du` starts with a number with more than four digits, `grep` will print that line, showing you the oversized subdirectory. The output of the above command might look like this:

```
72529  ./nwells/images/NASA_mars/
10218  ./nwells/database/archive/
21749  ./rsolomon/doc/HTML/
```

Running `du` on the root directory of a large system can take some time (and slow down everyone else's work). Any directory that contains thousands of files or hundreds of subdirectories requires some time for the `du` command to process.

To avoid drains on the system, consider using the `du` command in the middle of the night or at some other time when no one is using the system. Make it a practice to update a file containing the output of `du` each night. Then you can quickly search that file for overly large directories—directories that may require your attention (in the form of deleting or archiving files) if space becomes scarce.

In addition to using the `du` command, you can employ various graphical tools and system administration scripts that will automatically alert you to a file system that is approaching a threshold you specify. In Chapter 12, you will learn how to schedule tasks (such as running `du`) for the middle of the night, using simple programs called shell scripts.

## Creating New File Systems

As a Linux system grows, it will usually require additional storage space. As you will learn in Chapter 14, you can use archive systems to remove unused information and then store this data on compact disc, streaming tape, or other devices. Nevertheless, the amount of “live” storage needed often grows to exceed the administrator's original expectations. In fact, part of planning a Linux system in an organization is knowing in advance what steps will be taken when the system must be expanded. If these steps are outlined in advance, a system administrator is less likely to make choices that create obstacles to efficient system upgrades later on.

Adding a file system generally means adding a hard disk device to your system and making that hard disk available to Linux by formatting and mounting it. This process is similar to part of the Linux installation process. Because the installation utility takes care of some details described in this section, however, the steps described here may not be familiar to you. In this section you will learn how to make additional file systems stored on a hard disk, CD-ROM, or other device into active parts of your Linux directory structure.

You can install new file systems that are permanent (loaded each time you boot Linux) or temporary (loaded only occasionally as needed). File systems can be stored on a device with removable media (such as a cartridge) or fixed media (such as a hard disk). Some of the devices you might install on your computer are listed below. Data can be stored on any of these devices:

- CD-ROM drives
- CD writers
- DVD compact disc drives

- Tape drives
- Hard disks
- Iomega Zip and Jaz drives
- Other special removable cartridge devices

These devices can be connected to the computer's system board via an IDE interface, a SCSI interface, or by other proprietary expansion cards.



Linux can use the Network File System (NFS) to access file systems on other computers (that is, to access hard disks located on remote systems). It can also use NFS to mount such remote file systems as part of the local root file system. Many techniques described in this section apply to managing remote file systems. However, the details of using NFS are beyond the scope of this book.

The steps involved in installing a new hard disk in your Linux-based computer are beyond the scope of this book. You should consult your hardware manual for detailed guidelines. Once the hard disk is installed, you can use the Linux `fdisk` command to examine its partitions, creating new partitions for use by Linux if needed. Before any hard disk can be used as a native Linux file system, it must have Linux partitions defined. You learned about using the `fdisk` utility to create Linux partitions as you installed Linux. The `fdisk` utility can be used in the same way to prepare new devices installed on your system.



Devices with removable media such as Iomega Jaz disks and CD-ROM drives cannot have multiple partitions. With such devices, you can proceed directly to formatting the device for use by Linux.

Almost all file system devices use either an IDE or SCSI interface to communicate with your computer. Devices connected to these interfaces use standardized device names in Linux. Table 8-4 provides some examples designed to help you determine the correct name for a device. All IDE and SCSI devices are accessed via the `/dev` subdirectory.

**Table 8-4** Example Linux Device Names

Device	Description
<code>/dev/hda</code>	The first IDE device
<code>/dev/hdb</code>	The second IDE device
<code>/dev/hdc</code>	The third IDE device (often a CD-ROM drive)
<code>/dev/hda1</code>	The first partition on the first IDE device
<code>/dev/hdb3</code>	The third partition on the second IDE hard disk
<code>/dev/sda</code>	The first SCSI device
<code>/dev/sdb</code>	The second SCSI device
<code>/dev/sda4</code>	The fourth partition on the first SCSI hard disk
<code>/dev/sdc1</code>	The first partition on the third SCSI hard disk

For example, you can use the `fdisk` utility to set up partitions on a standard SCSI hard disk using this command:

```
/sbin/fdisk /dev/sdb
```

Once the `fdisk` utility has started, the `p` command in `fdisk` shows you the existing partition table; the `n` command starts the process of defining a new partition if you need to create a Linux partition on the new device. As you'll recall from installing Linux, the partition number is added to the device name. For example, if you create two partitions on the second SCSI hard disk, they are accessed as devices `/dev/sdb1` and `/dev/sdb2`.

If your device uses a special interface (not IDE or SCSI), Linux may already provide support for the device. In some cases you will need to contact the manufacturer to ask about Linux support. Alternatively, you can query a Linux mailing list or newsgroup to see if others have successfully used the device on a Linux system.

The extended file system 2 (`ext2`) is the default or native file system used by Linux. Once you have created the Linux partitions, you need to format the partition with the `ext2` file system. The command to create a new `ext2` file system is `mke2fs` (for *make ext2 file system*). This command formats the partition, erasing all information on it, and organizes space for data to be recorded so that the partition can be used by Linux. The command is as simple to use as adding the device name as a parameter:

```
/sbin/mke2fs /dev/sdb2
```

To format a hard disk partition, you may also use the `mkfs` program. This program requires a parameter to indicate the type of file system being created. The `mkfs` command simply starts the `mke2fs` command. This is an example of using the `mkfs` command to format a Linux file system:

```
/sbin/mkfs -t ext2 /dev/sdb2
```

When you format an `ext2` file system, you see many lines of output on the screen as the program lists all of the structure information that is being written to the device. In any case, formatting even a large hard drive is quite fast. On a Pentium system, using `mke2fs` on a 4 GB partition normally takes less than one minute to complete.



You can use a special command called `fdformat` to format floppy disks. This command is rarely used, however, because all floppy disks are preformatted; the man page provides detailed information.

## Mounting File Systems

After a new file system has been created (that is, formatted), it can be mounted as part of the Linux directory structure and accessed just like the existing Linux partitions. To add a new file system, you must create a directory as a mount point, which is a place in the directory structure where the file system can be accessed. Once you have created a directory to use as a mount point, use the `mount` command to activate the file system.

To create a directory, use the `mkdir` command and define the directory that you have chosen as an access point for the new file system. For example, if the new hard disk is intended as a document archive for your office, you could create a directory called `/archive`. If the new file system will be used for a database system and all applications are currently stored in the `/opt` directory, you might define the `/opt/db` directory as a mount point. The command would be:

```
mkdir /opt/db
```

After you create the directory, it is empty—it's just another directory on your root file system. The next step is to use the `mount` command to instruct Linux to access the new file system whenever you go to the `/opt/db` directory. The `mount` command can be complex. This is how the command would look using the example values used so far:

```
mount -t ext2 /dev/sdb2 /opt/db
```

This command says: mount a file system of type `ext2` (Linux native) located on the device `/dev/sdb2`, and make it accessible on the directory `/opt/db`.

Now when you go to the `/opt/db` directory, you'll find that it is no longer empty. Instead, it contains a directory with a strange-looking name: `lost+found`. The `lost+found` directory is placed in the beginning of all new `ext2` file systems. This directory is initially empty, but when you use disk-checking utilities (described in Chapter 9), files may be created in the `lost+found` directory. You will very rarely see anything in this directory, but don't delete it. The presence of the `lost+found` directory indicates that the new file system has been mounted successfully.

You can also use the `mount` command without any parameters to display a list of all mounted file systems, as shown earlier in this chapter. If you used the `mount` command alone in the previous example, the list of mounted file systems would include the `/dev/sdb2` device mounted on `/opt/db`.

The standard devices that are included when you install Linux generally have a mount point directory already created for them. For example, the floppy disk drive and CD-ROM drive are normally mounted to `/mnt/floppy` and `/mnt/cdrom`, respectively. These directories are created when Linux is installed. You can mount a CD-ROM by using this command, including the mount point as a parameter:

```
mount /mnt/cdrom
```

You can mount a floppy disk with this command:

```
mount /mnt/floppy
```

Why don't these commands include the information in the `mount` command given previously, such as a file system type and device name? You'll learn the answer shortly, in the section "Automating File System Mounting."

Once you have mounted a floppy disk or CD-ROM, you should not eject the disk or CD-ROM until you have unmounted the file system. If you do, Linux may not be able to access that device for a time.

## Unmounting File Systems

To unmount a file system, use the **umount** command with the device name or mount point. For example, to unmount a floppy disk, you might use this command (depending on the mount point defined on your Linux system):

```
umount /mnt/floppy
```

To unmount a CD-ROM, the command might be:

```
umount /mnt/cdrom
```

Similarly, to unmount a hard disk partition such as the one described in the previous section, the command would be:

```
umount /opt/db
```



The command name is **umount**, not **unmount**. If you see errors when you attempt to unmount a file system, look for an extra **n** in the command.

Keep in mind that a file system cannot be in use when you unmount it. If any users on the Linux system are working with a file on the file system, or if any user's current working directory is located on that file system, the **umount** command will fail, and it will indicate that the file system is busy. The Linux kernel stores information about each mounted file system that includes the number of files currently being accessed. All users must stop using files on the file system and change their current working directory to a location outside the file system before you can unmount that file system.

## Automating File System Mounting

As a system administrator, you'll want to automate everything that you can. Ideally, your systems should be as self-sustaining as possible, leaving you free for tasks that require new analysis and problem-solving skills. As you saw earlier, several types of file systems are mounted automatically when you start Linux. The new file systems that you create from additional hard disks or other devices can also be automatically mounted at boot time. Thus, you never need to enter the **mount** command after rebooting.

The key to automounting file systems is the **/etc/fstab** configuration file. This file contains a line for each file system that you want to have automounted when Linux boots. It

also contains a line for file systems that you want to mount later on without providing all of the file system information in the `mount` command. A typical default `fstab` configuration file is shown here:

```
/dev/hda3    /                ext2      defaults      1    1
/dev/hda4    /archive         ext2      defaults,noauto 1    0
/dev/hda2    swap            swap      defaults      0    0
/dev/fd0     /mnt/floppy      ext2      noauto        0    0
/dev/cdrom   /mnt/cdrom       iso9660   noauto,ro     0    0
none        /proc            proc      defaults      0    0
```

The fields of this configuration file, from left to right, are described in the following list:

- The device where the file system is located. Most of the devices in this example file are IDE hard disks; the floppy drive and CD-ROM drive have other device names; the `proc` file system has none.
- The mount point in the directory structure where the device will be accessed after being mounted. Each file system has a mount point directory except the `swap` file system, which is only used by the Linux kernel.
- The file system type. Most of the examples shown are `ext2`, the Linux native file system. `iso9660` is the CD-ROM standard; `proc` and `swap` are special file system types.
- Options that apply when this file system is mounted. You will learn more about these options later in this section.
- Whether the file system can respond to the `dump` command (this command is called `dumpe2fs` in Linux). A 1 in this field indicates that the `dumpe2fs` command can be used to print information on the structure of the file system. Only standard `ext2` hard disk partitions should have this field set to 1.
- The order used to check file systems when Linux is booted. Each time Linux starts, it checks the file systems in `fstab` before mounting them. The `root` file system (`/dev/hda3` above) should be numbered 1; other `ext2` file systems should be numbered 2. If 0 is used, the file system is not checked. All file systems that are not automounted can have 0 in this field.

The most powerful part of this configuration file is the options field. The options used to mount file systems are an effective way to increase security and ease system administration work. Key settings for the options field are described in Table 8-5.

Table 8-5 Important Option Field Settings

Mount option	Description
<b>async</b>	Specifies that all reads and writes to the file system should be asynchronous—in other words, that information will be buffered (stored in memory) to improve access speed.
<b>auto</b>	Specifies that the file system should be automatically mounted at boot time or when the <code>mount</code> command is used with the <code>-a</code> option.
<b>dev</b>	Designates the file system as a special device in the <code>/dev</code> directory.
<b>exec</b>	Permits programs stored on the file system to be executed.
<b>noauto</b>	Indicates that this file system should not be automatically mounted. Instead, the file system must be mounted by an explicit <code>mount</code> command.
<b>noexec</b>	Indicates that programs stored on the file system cannot be executed.
<b>nouser</b>	Specifies that no regular users can mount the file system; instead, only <code>root</code> can mount it.
<b>ro</b>	Mounts the file system as read-only, which means no data can be written to it.
<b>rw</b>	Mounts the file system as read/write—the standard mode in which data can be written to the file system.
<b>suid</b>	Allows special user ID permissions to be used on this file system.
<b>user</b>	Allows a regular user to mount the file system. This is useful if you are running a desktop Linux system and don't want to switch to the <code>root</code> user account to mount a floppy or CD-ROM.
<b>users</b>	Functions the same as the <code>user</code> option except that any user can unmount the device.
<b>defaults</b>	Includes the options <code>rw</code> , <code>suid</code> , <code>dev</code> , <code>exec</code> , <code>auto</code> , <code>nouser</code> , and <code>async</code> .



When removable media such as a CD-ROM is mounted, the Eject function of the drive is disabled (until the device is unmounted). Keep in mind that this is not true of floppy disk drives, which means it is possible for a user to eject a floppy disk while it is still mounted. This can cause problems, resulting in lost data. For example, files will still be marked as open even though they cannot be accessed. Although the `user` option in Table 8-5 is appropriate for some situations, it may create the problem noted here if users are not aware of the need to unmount devices before ejecting media.

Additional (less commonly used) options are described in the manual page for the `mount` command. Two important additional points about the options in the `fstab` file must be mentioned. First, the last options listed on each line of the `fstab` file override any earlier

options used on the same line. For example, if the options list includes `defaults`, `user`, the `user` option overrides the `nouser` option that is part of `defaults`. Second, the options can be added to the `mount` command by using the `-o` parameter. For example:

```
mount -t ext2 -o defaults /dev/sdb2 /opt/db
```

Let's walk through two examples of how you might add a line to the `fstab` file to automate file system mounting. As a first example, suppose you had set up a large SCSI hard disk to hold a database application. You want the database file system to be mounted automatically when you start Linux. Users should not be able to run programs or modify the file system configuration. For this situation, the `fstab` line might look like:

```
/dev/sdb2 /opt/db ext2 defaults,noexec,nosuid 1 2
```

As a second example, suppose you are using Linux as your desktop workstation and have installed a new DVD drive. The device shouldn't be mounted at boot time because you don't normally keep a DVD disk in the drive, but you want to be able to mount the device without changing to the `root` user account. You also want to protect any writeable DVD disks from having any data damaged. You might use this line in the `fstab` file:

```
/dev/hdd /mnt/dvd iso9660 ro,noauto,user 0 0
```

Once you have the `fstab` file set up, you can use the `mount` command with only the device name or mount point. The `mount` command looks in the `fstab` file for all of the additional information needed to mount the file system. For example, the CD-ROM device is normally configured in `fstab` after a Linux installation. The CD-ROM can thus be mounted with this command:

```
mount /mnt/cdrom
```

No additional information is needed on the command line because `mount` retrieves everything else from the `fstab` file. If you use the `mount` command without sufficient information (and the information is not contained in the `fstab` file), an error message is displayed and the file system is not mounted.

The `mount` command does only minimal checking when a file system is first mounted. This means that Linux may allow you to proceed with a `mount` operation that appears to provide access to a file system, when in fact the file system is not supported. As you begin using the file system, Linux may discover the problem, and display an error message such as `not a valid block device`.

Nevertheless, Linux supports many different file systems. If you have partitions on your computer's hard disks that use any of the file system types listed in Table 8-6, you can access them directly from Linux by mounting them. Support for each of these file system types may be built into your Linux kernel, or you may need to load a kernel module to enable support for a specific file system type. The module you load provides back-end support in the kernel for a specific file system type.

**Table 8-6** File System Types Supported by Linux

File system	Description
<b>ext2</b>	The native Linux file system type.
<b>minix</b>	File system used by the Minix operating system.
<b>msdos</b>	File system used by DOS and older Microsoft Windows systems. Note that this file system is also called the FAT file system, but the <code>msdos</code> name must be used with the <code>mount</code> command and with the <code>fstab</code> file.
<b>hpfs</b>	The High Performance File System used by OS/2.
<b>iso9660</b>	File system used on CD-ROMs.
<b>nfs</b>	The Network File System, used to allow networked computers to access remote hard disks as part of a local directory structure.
<b>smbfs</b>	File system used to mount SMB network devices such as networked Windows computers. This file system type is part of the Samba suite. (For more information, see <a href="http://www.samba.org">www.samba.org</a> .)
<b>vfat</b>	File system used by Windows 98; also known as the FAT32 file system. Provides long filename support when reading Windows partitions from Linux.
<b>ntfs</b>	File system used by Windows NT. This file system requires kernel configuration that is normally not enabled by default.
<b>sysv</b>	A standard UNIX System V file system.
<b>qnx4</b>	File system used by the QNX operating system.
<b>coherent</b>	File system used by the Coherent UNIX operating system.
<b>ufs</b>	A UNIX file system type.
<b>xenix</b>	File system used by the Xenix operating system, a variant of UNIX.



The `msdos` file system type provides access to DOS or Windows file systems. Because these file system types do not provide the same features as more robust file systems such as `ext2`, the back-end to the `msdos` file system within the kernel maps features between `msdos` file systems and Linux. For example, the end-of-line characters are different in DOS files and Linux files. Linux also has additional file attributes compared to DOS. The `msdos` file system back-end maps between these differences to make `msdos` files usable in Linux.

## Managing Swap Space

As mentioned in Table 8-3, the `swap` file system (often called the swap space) is a special file system type used by the Linux kernel for virtual memory. **Virtual memory** is a kind of memory that is used like standard RAM; however, information in virtual memory is stored on a hard disk instead. The swap file system is set up during Linux installation and activated (via the `fstab` file) each time Linux boots.

The swap space is normally a separate hard disk partition. This allows the most efficient access by the Linux kernel. For systems without an available partition for the swap space, a file on the regular `ext2` file system can be designated as the swap space instead. However, this technique degrades performance and should only be implemented on a system set up for testing Linux.

In some Linux systems, the maximum size of a swap partition is 128 MB. With today's large Linux systems, this may be insufficient. On some new Linux systems (running Linux 2.2 kernels or later versions), a swap partition can be 2 GB or larger, depending on the platform on which Linux is running.

You use the `mkswap` command to create a swap partition. This is similar to using the `mke2fs` command to create an `ext2`-formatted partition. If you have set up a swap partition using `fdisk` or another partitioning tool, the command would be something like this (depending on the partition you need to format as swap space):

```
mkswap /dev/hda2
```

Once you have formatted a partition as swap space, you need to add a line to the `fstab` file that tells Linux to use the swap automatically. The example file shown previously includes this line:

```
/dev/hda2          swap    swap    defaults    0 0
```

Swap space is activated by system initialization scripts during the system start-up phase. The `swapon` command is used in these scripts to activate swap space.

With large or busy systems, the swap partition should be located on a separate hard disk from the root partition or other key data partitions. If the system is large enough, a separate hard disk might even be used just for swap space. By placing the swap space on a separate hard disk, you ensure that information can be read from and written to the root or other data partitions without interfering with the kernel's efforts to move data to and from the swap area. The two (or more) hard disks can act in parallel instead of completing one operation after another. For the same reason, some Linux system administrators set up multiple swap partitions to take advantage of parallel hard disk accesses. Multiple swap partitions were sometimes needed in older versions of Linux to overcome the 128 MB limit, but they may still be used to achieve speed increases by having the kernel access multiple hard disks in parallel. Linux supports up to eight distinct swap areas. Each one can be created using the `mkswap` command and included in the `fstab` file to be activated at boot time.

The status of swap space (or virtual memory) can be viewed using the **free** command. Following is a typical example of output from this command:

	total	used	free	shared	buffers	cached
Mem:	30820	28768	2052	16852	11860	7104
-/+		9804	21016			
buffers/ cache:						
Swap:	130748	1248	129500			

Note the **total** and **free** columns. **Mem** refers to RAM memory; **Swap** refers to swap space. On the small system shown here, a total of about 32 MB of RAM and 130 MB of swap space are available on the system. Very little of the swap space is used, and most of the RAM is free also (21 MB). As the system becomes busy, the free memory will drop to zero, and the swap space will be used.

Because swap space is located on a hard disk, it is significantly slower to access than the RAM on your system board. Because of this, you may at first want to avoid using it by installing enough RAM so that the Linux kernel never needs to save information in the swap space. In truth, both RAM and swap space are useful. A typical system should have at least as much swap space as RAM. The reason for this is that the Linux kernel will attempt to cache (store) in RAM as many files and programs as possible in order to increase the system speed. But if many users on a system are working with many programs, it's likely that not all programs will be active at the same time. Rather than have these programs use RAM, they can be placed in the swap space while they are inactive. On a well-tuned system, the kernel can quickly bring these programs back into memory when a user needs them—so quickly that most users won't notice any delay.

Another reason to make wise use of swap space is cost. Swap space (that is, hard disk space) is so much cheaper than RAM that in many situations adding more swap space is more cost effective than adding RAM.

However, while swap space is an important part of your system, its advantages diminish on systems with insufficient RAM. On systems with too little RAM, the swap space will be overused. This means that a single program might be moved to the swap space and back into system memory several times per second, as it competes with other programs for processor time. The time required to move information to and from the swap space greatly reduces the efficiency of the system. This problem is called **thrashing**. To solve this problem you need to reduce the system load or add more system RAM.

To see detailed information about how the swap space is used, try the **vmstat** command (for *virtual memory statistics*). The output of this command is cryptic and requires careful study of the relevant manual page. (See also Chapter 10.) The output of **vmstat** provides information on which processes are using swap space, how much space they are using, what is waiting to happen when RAM is available, and so forth.

If the Linux kernel runs out of swap space (in which case it runs out of memory), the kernel may crash. This very rarely happens, but the possibility is reason enough for monitoring swap space use.

## SIMPLE TASK MANAGEMENT

In Chapter 7 you learned that the **ps** command can be used to list the processes running on Linux. In the sections that follow, you will learn more about how to control those processes. Additional detail on managing the CPU load via other utilities is provided in Chapter 10.

### Job Control in the Shell

Often you will want to start multiple programs from the command line at the same time. These might include an editor, a script you have written, an e-mail reader, and perhaps other programs as well. Although Linux can easily run many applications (processes) at the same time, you must learn to control all of these processes from a single command line. Most of the tools for doing this are part of the **shell**, or command interpreter. Because the default shell for Linux is **bash**, commands described in this section apply to the **bash** shell. Other shells such as **ksh**, **csch**, **tcsh**, and **zsh** support similar features.

You may choose to work in command-line windows within a graphical display, in which case you can open multiple windows to start multiple programs. You may also use the virtual consoles described in the next section. On occasion, however, you will need to manage multiple programs from a single command line. This discussion also helps you understand how processes are managed and how the shell operates.

### Processes

When you start a program, that program takes control of the command line where you are working. For example, if you enter the command

```
man ps
```

the man page appears, and you no longer see a prompt where you can enter additional commands. Some commands don't display screen output like the man page viewer, but they still leave you without an active prompt to enter additional commands.

If you type an ampersand after the name of a command, the shell places the process in the background—in other words, the process continues to run, but it doesn't control the command line. You can then start another command immediately. Multiple processes started from a single shell are called jobs.

### Jobs

A job is simply a process that is associated with a single shell or command-line environment. You can use the command **jobs** to list all jobs or processes that are running from the current shell.

Use the Ctrl+Z key combination to suspend a job. This key combination is useful if you have started a command that is controlling the command line and you want to interrupt it so that you can use the **bg** command to place it in the background. A suspended job is not ended, but it stops running normally. It waits for further instructions before beginning normal

execution again. You can use the `jobs` command to see which processes are currently suspended, as in the following output:

```
$ jobs
[1]+  Stopped (signal)          top
```

The output of the `jobs` command shown above includes a job number at the left of the line (1 in the output above). The process ID number is not shown by the `jobs` command. When a job is suspended, you can either place it in the background (restart it without displaying output to the current console) or place it in the foreground (allow it to take control of the screen again). Figure 8-4 shows how a single command-line window can start and manage multiple processes (or jobs).

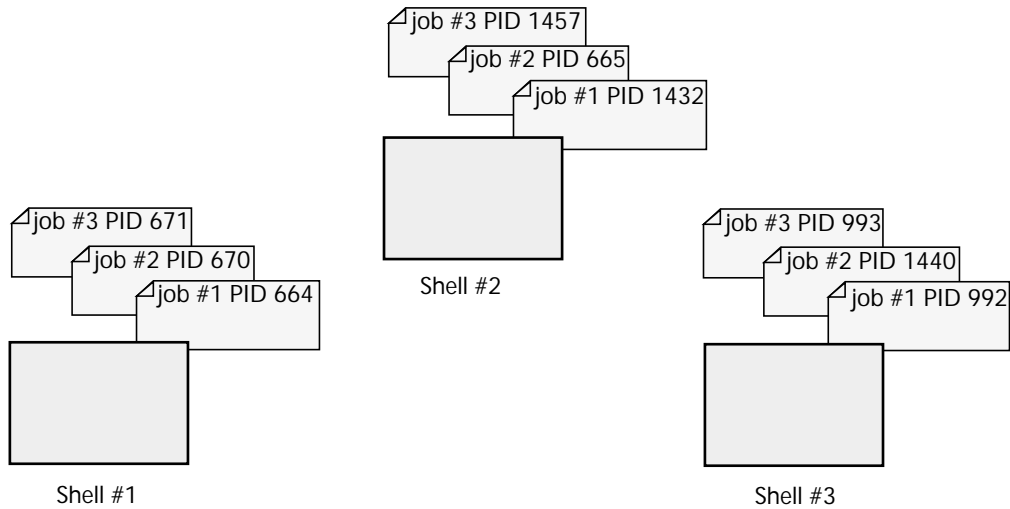


Figure 8-4 Multiple jobs running from one shell

The commands used to place a job in the background or foreground after it has been started are `bg` and `fg`. To use the `bg` or `fg` command, you must know the job number assigned by the current shell. You can find this number by using the `jobs` command.

The following steps show how the `bg` and `fg` commands work:

1. At any Linux command line, enter the command `man ls`. The manual page for the `ls` command appears.
2. Press **Ctrl+Z** and then **Enter**. You see the following message (the number at left may be different on your system):
 

```
[4]+  Stopped                  man ls
```
3. Enter the command `man ps`. The manual page for the `ps` command appears.
4. Press **Ctrl+Z** and then **Enter**. You see the following message (the number at left may be different on your system):
 

```
[5]+  Stopped                  man ps
```

5. Enter the command `jobs`. You see output like the following:

```
[4]+  Stopped                  man ls
[5]+  Stopped                  man ps
```

6. Enter the command `fg %4`. (Use the number at the left of the first output line from the `jobs` command—on your system it may not be 4.) The `ls` manual page appears again.
7. Press `q` to end the `man ls` command.
8. Enter the `jobs` command again. You see that the `man ls` command is no longer listed.



Some commands (including the `man` command) are only used to display information. Placing a command-line `man` in the background with the `bg` command automatically suspends the command. It only runs in the foreground.

8

You can also use the process ID number in the `fg` or `bg` command. Just use the number without the percentage sign. For example, suppose a process you have started is job number 3 in the current shell and has a PID of 725. You can bring the process to the foreground with either of these commands:

```
fg %3
```

```
fg 725
```

## Using Virtual Consoles

As you learned in Chapter 4, you can open multiple command-line windows within a graphical environment. When you are not working in a graphical environment, you can use virtual consoles in Linux to start multiple command-line sessions at the same time. A **virtual console** is a separate login screen that you access by pressing a combination of keys on your keyboard. A virtual console allows you to start several separate login sessions in Linux from the same computer.



Networked Linux systems allow many users to log in using a network connection. Virtual consoles provide the same type of login functionality without a network connection.

When the graphical mode is not active, you access a new virtual console by pressing `Alt+F2`. This displays a new login prompt, where you can log in using any valid username and password. Any commands that you start from this virtual console run independently of those on other virtual consoles. Each console starts a separate copy of the `bash` shell, so the `jobs` command will only list jobs started in one virtual console, even if you have logged in using the same username.

To switch back to your first virtual console, press Alt+F1. Most Linux systems have six virtual consoles. They can be accessed by pressing Alt+F1 for the first virtual console, Alt+F2 for the second, and so on, to Alt+F6 for the sixth virtual console. A graphical system normally appears on a seventh virtual console, which you can display by pressing Alt+F7.

When the graphical environment is running, you can switch to the nongraphical virtual consoles by pressing Ctrl+Alt+F1 for the first virtual console, and so on, to Ctrl+Alt+F6 for the sixth virtual console.

## Learning About Processes

The `ps` command that was introduced in Chapter 7 includes many additional options to help you learn about what is happening on your Linux system from moment to moment. A simple `ps` command shows you only the commands that you have started in the current command-line environment (also called the current terminal):

```
$ ps
PID TTY          TIME CMD
 576 tty1        00:00:00 login
 584 tty1        00:00:00 bash
 946 tty1        00:00:00 top
 951 tty1        00:00:00 ps
```

Although this is useful, it doesn't provide much more information than the `jobs` command. As a system administrator, you need access to the details provided by the `ps` command options. For instance, the `a` and `x` options show you the processes started by all users, as well as those that were started by the system at boot time (or other processes that have no controlling `tty` where they were started).

This is a much longer list than that provided by the `ps` command without any options, and it includes all of the system-level daemons that are running in the background as you work on Linux. For example, the `login` command running on other virtual consoles, the Web server (called `httpd`), the system logging daemon, and possibly an e-mail server will all appear in the list. By adding the `u` option to the `ps` command, you can also see information about how each process is using your Linux system. This command is shown here with its output sent to the `less` command. By using the `less` command, you can use the Page Up and Page Down keys to view the many lines of output. The first few lines of output are shown after the command:

```
$ ps aux | less
USER      PID %CPU %MEM    SIZE   RSS TTY  STAT  START   TIME COMMAND
bin        381  0.0  0.9    840    300 ?    S     13:32   0:00 rpc.portmap
daemon    451  0.0  1.9   1156    596 ?    S     13:32   0:00 lpd
daemon    471  0.0  1.0    828    324 ?    S     13:32   0:00 atd
nobody    845  0.0  2.5   1384    784 ?    S     13:32   0:00 httpd -f
nobody    846  0.0  2.5   1384    784 ?    S     13:32   0:00 httpd -f
root       1  0.0  1.0    828    332 ?    S     13:31   0:04 init
root       2  0.0  0.0      0      0 ?    SW    13:31   0:00 (kflushd)
root       3  0.0  0.0      0      0 ?    SW    13:31   0:00 (kpiod)
```

In the first lines of the output shown above, you see column headings indicating which user started the process, the percentage of CPU time and memory used by the process, the terminal that the process is running on, status, and other information. The most important part of this information is the process ID number, or PID. With this number, you can control the process using other Linux commands.

You can use the `f` option to display the relationship between different processes, showing which process started which other processes. The following command uses the `f` option along with the `a` and `x` options, in order to display all the processes on the system (the output has been shortened to save space here, however):

```
$ ps axf
PID TTY          STAT TIME COMMAND
   1 ?            S    0:04 init
   2 ?            SW   0:00 [kflushd]
   3 ?            SW   0:00 [kpiod]
  535          S    0:00 sendmail: accepting connections: p
  550 ?          S    0:00 gpm -t ps/2
  564 ?          S    0:00 httpd
  568 ?          S    0:00 \_ httpd
  571 ?          S    0:00 \_ httpd
  577 ?          S    0:00 \_ httpd
  594 ?          S    0:00 xfs
  638 tty2       S    0:00 login -- root
  664 tty2       S    0:00 \_ -bash
  676 tty2       T    0:00 \_ \_ man ls
  677 tty2       T    0:00 \_ \_ \_ sh -c /bin/gunzip
  678 tty2       T    0:00 \_ \_ \_ \_ /bin/gunzip
  679 tty2       T    0:00 \_ \_ \_ \_ /usr/bin/less -is
  680 tty2       T    0:00 \_ \_ top
  686 tty2       R    0:00 \_ \_ ps axf
  639 tty3       S    0:00 /sbin/mingetty tty3
  642 tty6       S    0:00 /sbin/mingetty tty6
  644 ?          S    0:00 update (bdflush)
```

As you can see, the processes are presented in a tree diagram. This output shows which processes were started by other processes. For example, process ID (PID) 638 (see the left column of the output above) is the `login` command, where a user has logged in as `root`. The `login` process started a `bash` shell (the next line in the output, process 664). The `root` user started several commands within the shell, including `man ls` (PID 676), `top` (PID 680), and the `ps` command (PID 686). The `man` command started other commands to uncompress the manual page file. Many processes were started by the Linux kernel when the system was booted. These processes appear without any tree structure.

Before you can successfully control process operation and manage your Linux CPU and memory resources, you need to understand how processes are related to one another. Each process has a parent process—that is, the process that started it. A parent process can have many child processes. The first process started on a Linux system is called `init`. This process is the parent to all processes and has a PID number of 1.

## Controlling Processes

You can use the `jobs`, `fg`, and `bg` commands to control processes (jobs) that were started within a single shell. By using the `kill` command, you can control all processes on the system. The name of this command is a little unfortunate. Although it is often used to kill, or end, processes, it actually is used to send signals to processes. Some of those signals end the process; others serve different purposes, such as suspending a process or causing a program to reread its configuration file.



You must be logged in as the `root` user to control processes that you didn't start.

**Signals** are messages that are sent to a process. The full list of signals contains about 30 messages, but most of these are not used regularly. Each signal has a name and a number associated with it. To see a cryptic list of all the signals, use the `kill` command with the `-l` option (for list), as follows:

```
kill -l
```

When writing a program, the developer decides which signals the program will respond to. Some programs only respond to one or two signals. Others may respond to more signals, depending on the purpose of the program. For example, a program designed to control your computer in the event of a power failure will respond to the signal from a power supply indicating that the main power is out. Other programs wouldn't respond to this signal.

Almost all programs respond to the `SIGTERM` signal (signal number 15). This signal requests that the program end. Another special signal is `SIGKILL` (signal number 9). The `SIGKILL` signal is not handled by the program itself. Instead, if you send a `SIGKILL` using the `kill` command, the Linux kernel shuts down the process automatically. Any unsaved data in a program will be lost when the `SIGKILL` signal is used to end the process. As a rule, you should use the `SIGTERM` signal (rather than the `SIGKILL` signal) to shut down processes, because `SIGTERM` requests that a program close itself, giving the program a chance to clean up its work, close any open files, and so forth, before ending. When you use a `SIGKILL`, the process is cut off before it can do any of those things. However, `SIGKILL` is very useful when a process is not responding to the `SIGTERM` signal.

To see how the `kill` command is used, suppose a user on your Linux system had started a program called `myeditor`. The program appears to have stopped working but is still running in the background. You would use the following `ps` command to see that state of the process:

```
ps ax | grep myeditor
```

The single-line output of this command includes only the process for the `myeditor` program with the PID number for the process. Using this information, you can send a signal to the process (because the user in question started the process, he could also use the `kill` command to send the signal):

```
kill -15 1482
```

By sending the SIGTERM signal in this way, you send a request to the `myeditor` program to close. The command could also be written using the name of the signal:

```
kill -SIGTERM 1482
```

If the program does not respond to the request to terminate (you still see it in the list of processes from the `ps` command), you can send a SIGKILL signal that will cause the Linux kernel to end the process immediately:

```
kill -9 1482
```



Before you can send a process a signal, you must obtain the correct process ID for the process by using the `ps` command.

A special form of the `kill` command is the command named `killall`. This command is used to send a signal to all processes that were started by commands of the same name. This is useful when a program is starting copies of itself faster than you can locate the PIDs and use `kill` to end them. If the `myeditor` program were doing this, you could use this command:

```
killall -9 myeditor
```

Always be careful using `killall`. If you are running multiple programs of the same name and only one needs to be sent a signal, do not use `killall` because it will end all of the processes. Instead, determine the PID of the specific copy of the program that needs to be ended. Then use the `kill` command rather than `killall`.

This section has provided only an introduction to the concepts and commands used to control processes in Linux. In Chapter 10 you will learn how to manage the load on your Linux system by allocating time and disk space to processes and tracking how busy your Linux system is to see if additional hardware resources may be required to handle the load.

## CHAPTER SUMMARY

- ❑ A Linux system administrator must perform some basic tasks regularly to keep the system running smoothly. These tasks include user management, file system management, and process management.
- ❑ User management involves defining user environments, creating user accounts, and managing modifications to those accounts as required by the system or requested by the user. Maintaining user accounts is an important part of system security.
- ❑ File system management provides stability and performance for your Linux system. Each file system must be tracked to see how it is being used and when new hardware resources need to be added to handle the system's load. Working with file systems must be done carefully to avoid damaging data.
- ❑ To manage processes, you use commands to examine all the processes on the Linux system and send various signals to control how those processes behave. The `root` user can control all processes, suspending or stopping any process.

---

## KEY TERMS

- alias** — Command used to create a text substitution in a command-line shell, effectively giving any Linux command a new name.
- bash** — Short for Bourne again shell. It is the default command-line interpreter for Linux.
- bashrc** — Configuration file containing commands that are executed each time a user starts a new command-line environment.
- bg** — Command used to place a job (process) in the background (either by suspending it or by preventing its output from appearing in the current shell's terminal window), thus allowing the shell prompt to become active again.
- C shell** — A shell designed for ease of use more than for programming features.
- cat** — Command used to dump the contents of a file to STDOUT.
- command interpreter** — (More commonly called a shell in Linux.) A command-line environment in which a user can enter commands to be launched.
- default shell** — The default command-line interpreter used in most Linux systems (**bash**).
- df** — Short for display file systems. Command used to display file system summary information such as device, mount point, percentage used, and total capacity.
- du** — Short for disk usage. Command used to display disk space used by a directory and each of its subdirectories.
- environment variables** — Set of named values (name-value pairs) that provide information to programs running in a user's environment.
- /etc/fstab** — Configuration file that contains a file system table with devices, mount points, file system types, and options. Used by the mount command.
- /etc/group** — Configuration file in which group information (group names and membership lists) is stored.
- /etc/passwd** — Configuration file in which user account information is stored.
- /etc/shadow** — Configuration file in which encrypted user passwords and password configuration data are stored.
- /etc/skel** — Directory containing files that will be used to populate a new user's home directory at the time it is created.
- fdformat** — Command used to format a floppy disk.
- fg** — Command used to bring a job (process) running in a shell to the foreground so that the job controls the shell's terminal window.
- file system** — A collection of data, normally stored on a device such as a hard disk partition, which can be accessed in Linux via the directory structure.
- group** — A collection of user accounts that can be granted access to the system collectively.
- groupadd** — Command used to add a new group to a Linux system.
- init** — Linux process that initiates other key processes as the system is booting.
- jobs** — Command used to list jobs (processes) started in the current shell environment.
- kill** — Command used to send signals to processes, often to end them via a SIGTERM or SIGKILL signal.

- Korn shell** — A revision of the `bash` (or Bourne) shell that is popular on some UNIX systems and available in Linux as the Public Domain Korn shell, `pdksh`.
- mke2fs** — Command used to format a device such as a hard disk partition with an `ext2` file system.
- mkfs** — Command used to format devices using various file system types. The `ext2` default type for Linux file systems can be indicated as an option. *See also* `mke2fs`.
- mkswap** — Command used to format a partition as a swap space for the Linux kernel.
- mount** — Command used to make a logical or physical device available as a file system in the Linux directory structure.
- mount point** — The place or path in the Linux directory structure where a file system is accessed.
- ps** — Command used to obtain detailed information about processes running on Linux.
- root** — Superuser account name in Linux.
- Shadow Password system** — Security system used to restrict access to encrypted password text.
- shell** — A command-line interpreter, providing a command-line interface.
- signal** — A message (one of a fixed set determined by the Linux kernel) that can be sent to any process and responded to according to how that program is written.
- su** — (Short for substitute user.) Command used to take on the identity of a different user account.
- superuser** — The root user account, which has supervisory privileges throughout the Linux system.
- swapon** — Command used by Linux initialization scripts to activate the swap partition defined in the `/etc/fstab` file.
- symbolic link** — A pointer in the file system to another file.
- thrashing** — Excessive movement of processes between RAM and swap space, resulting in reduced system performance and excessive wear on the hard disk.
- touch** — Command used to create an empty file or to update the access time of an existing file.
- umount** — Command used to unmount a file system that is accessible as part of the Linux directory structure.
- User Private Group** — Security system that creates a new group containing one user when that user is first created.
- useradd** — Command used to create (add) a new user account in Linux.
- usermod** — Command used to modify or update an existing user account.
- virtual memory** — Memory available to the Linux kernel for running programs but which is actually located on a hard disk. Data that the Linux kernel stores in virtual memory is placed in the swap file system, or swap space.
- wheel** — Special system administrative group, not used officially in Linux.

---

## REVIEW QUESTIONS

1. Name two reasons why you shouldn't log in as `root` unless you are doing system administration work.
2. A user's primary group can be a User Private Group. True or False?
3. The `/etc/passwd` file does *not* contain which of the following fields:
  - a. The name of the user account
  - b. The file privileges for the user
  - c. The user's default shell
  - d. A UID and GID for the user
4. Explain the meaning of this line in the `/etc/group` file:  
`webmasters:x:710:rthomas,cyang`
5. To create or change a password on any user account, the following is used:
  - a. The `useradd` utility
  - b. The file `/etc/shadow` with a text editor
  - c. The `passwd` command
  - d. The UID and GID of the user
6. The `useradd` command can be used to modify or update account information. True or False?
7. When you add a file to the `/etc/skel` directory, the file is automatically added to the home directory of all existing users. True or False?
8. A \_\_\_\_\_ defines a string of text to be substituted whenever another string of text is used on the command line.
  - a. link
  - b. substitution string
  - c. alias
  - d. symbolic link
9. The Linux command used to format a Linux `ext2` hard disk partition is:
  - a. `mke2fs`
  - b. `fdisk`
  - c. `fsck`
  - d. Linux does not use formatted partitions
10. List two ways in which regular user accounts differ from nonstandard user accounts that are used only by Linux programs.
11. If you enter a new password for a user account that can be easily guessed, the message `BAD PASSWORD` appears and the password is not updated. True or False?

12. Environment variables are used by nearly all users to:
  - a. Provide system or user information to the programs that the user executes
  - b. Make it easier to use DOS commands in a Linux environment
  - c. Track system resources used by each user
  - d. Record environment information as users work with system administration tools
13. Describe one simple method of temporarily disabling a user account without using a graphical utility.
14. A mounted file system is one that:
  - a. Has been included as part of the Linux directory structure
  - b. Has been correctly formatted for use in Linux
  - c. Allows any user to run programs located on it
  - d. Includes at least a `root` user account
15. Describe the value of multiple virtual consoles in Linux.
16. The Shadow Password system enhances Linux security by:
  - a. Validating members of the `wheel` group as they log in
  - b. Hiding encrypted passwords in a file that only `root` can read
  - c. Checking that new passwords entered for users are not easily guessed
  - d. Stopping unauthorized users from accessing the `root` account
17. Which of the following is *not* a good way to create a new user account:
  - a. Use the `useradd` command.
  - b. Use the `LinuxConf` utility in Red Hat Linux.
  - c. Add a line to `/etc/passwd` with appropriate information.
  - d. Start the Shadow Password system.
18. Which of the following is *not* a valid reason to use aliases in a user's environment settings:
  - a. They save processing time as commands are executed.
  - b. They protect against accidental erasure of files.
  - c. They correct typographical errors as you work.
  - d. They make non-Linux commands behave in a familiar way.
19. The `df` utility provides information about which one of the following:
  - a. Which users have mounted the file system
  - b. The virtual memory usage as stored on all mounted file systems
  - c. File system capacity, device name, and percentage used status
  - d. Per-directory usage and file system mount point

20. Describe the actions of the `defaults` options in a configuration line of the `/etc/fstab` file.
21. Which of the following is a valid `alias` command?
  - a. `alias cp copy`
  - b. `alias copy=cp`
  - c. `alias DOS COPY to Linux cp`
  - d. `export alias=copy,cp`
22. Describe the key advantage to having swap space located on a hard disk separate from the root Linux partition.
23. List two reasons why a user account might need to be disabled.
24. If you attempt to unmount a mounted file system and receive an error message, the most likely cause is:
  - a. The file system was not mounted correctly in the first place.
  - b. The `df` command is in the process of computing file system statistics.
  - c. An error on the physical media that Linux cannot interpret.
  - d. One or more users are working in the file system.
25. By starting multiple jobs in one command-line session, you can
  - a. Conserve resources for each process you start
  - b. Prevent the swap space from thrashing
  - c. Manage those jobs with the `jobs`, `fg`, and `bg` commands
  - d. Kill any unneeded process quickly
26. Signals are used by the `kill` command to manage processes. True or False?
27. Describe at least four fields of information provided by the command `ps auxf`.

---

## HANDS-ON PROJECTS



### Project 8-1

In this project you will create a new group and a new user account using the `groupadd` and `useradd` commands. Then you will update the user information on that account using `usermod`. To complete this project you need to have Linux installed. You should be at a Linux command-line prompt and have `root` access in order to perform these steps.

1. Enter the following command: `groupadd webmasters`. This creates a new group named `webmasters`.
2. Enter the following command: `cat /etc/group`. This displays the group file, with the new group in the last line of the output.

3. Enter the following `useradd` command: `useradd -g webmasters -e 06/30/01 -c "Hailey Mendez" hmendez`. This creates a new user account with an expiration date, full name in the comment field, and primary group assignment.
4. Enter the following command: `cat /etc/passwd`. This displays the user file (`/etc/passwd`), with the new user account information in the last line of the output.
5. Determine whether a home directory was created by the command in Step 3 by entering the following command: `ls /home/hmendez`. If you see a list of files, the home directory was created in Step 3; if you see an error message, the home directory was not created. If the home directory was not created, create it with this command: `mkdir /home/hmendez`.
6. Set a password for the new user account using this command: `passwd hmendez`.
7. Enter a new password twice as prompted.
8. Change to the new user account using the following substitute user command: `su - hmendez`. (Note that no password is required because you are logged in as `root`.)
9. Enter the following command: `alias`. This displays the aliases that are in effect for the new user. (Note that this command displays your system's defaults, unless you have changed the configuration files.)
10. Enter the following command: `exit`. As a result, you exit from acting as the new user and return to the `root` user account.
11. Now suppose you have been asked to change the login shell used by this user. Change the shell with the following `usermod` command: `usermod -s /bin/tcsh hmendez`.
12. Enter the following command: `cat /etc/passwd`. This displays the user file, where you can see the effect of the `usermod` command. Specifically, note `/bin/tcsh` in the last field of the line defining the `hmendez` account.



## Project 8-2

In this activity you will explore how a signal sent with the `kill` command affects a Web server running on your Linux system. Most Linux systems have an `httpd` daemon (a Web server) that runs automatically after installation. You should have a completely installed Linux system to complete this task. Ideally, the Linux system should have a Web server installed by default. If you have not selected this installation option, you may be able to start a Web server using the command `httpd`. Log in to Linux and open a command-line window to complete the steps that follow.

1. Filter the `ps` command output through the `grep` command as follows: `ps aux | grep httpd`. This lists all of the Web server processes running on your computer.

2. Look at the output and notice the `owner` field on the far left (see Figure 8-5). Why is one copy of `httpd` owned by `root` and the others owned by a user named `nobody`?

```

[root@brighton root]# ps aux | grep httpd
nobody    1037   0.0  2.4 1384   756 ?    S    13:57   0:00 httpd -f /etc/httpd/a
nobody    1038   0.0  2.4 1384   756 ?    S    13:57   0:00 httpd -f /etc/httpd/a
root       860   0.0  2.3 1360   740 ?    S    13:28   0:00 httpd -f /etc/httpd/a
root      1082   0.0  0.9   916   308 p0    S    14:47   0:00 grep httpd
[root@brighton root]#

```

Figure 8-5 Output from `ps aux | grep httpd`

3. Use the same command again with the `f` option, as follows: `ps auxf | grep httpd`. This displays the parent-child relationship of the Web servers. You can see that the copy owned by `root` started the other copies.
4. Note the start-time field (third field from the right) for each of the Web server daemons.
5. Note the PID of the Web server daemon owned by `root`. (The PID is the number nearest the far left column.) (The following steps will use the number 515, but you should replace this number with the PID that appears when you run the `ps` command on your system.)
6. Send a restart signal to the parent Web server process using the following `kill` command: `kill -HUP 515`. This signal causes the Web server to reread its configuration and restart all child processes.
7. Use the following `ps` command again: `ps aux | grep httpd`. Notice the start-time field. All of the processes owned by `nobody` (the child processes of the main Web server daemon) have been restarted and have a new start time. The main process owned by `root` does not have a new start time.

## CASE PROJECTS

1. You have been asked to help design the file systems to be used for a large new Linux system in your research lab. The Linux system will support about 50 researchers who will log in each day to run scientific applications and access relevant Internet resources. The Linux system will also act as a news server, receiving about 1 GB of newsgroup messages each day over the Internet. A large database application runs on the Linux server to provide research data. As the system administrator, you expect to upgrade the Linux operating system about once per year; you will also maintain complete backups on a writeable CD-ROM drive.

Design a structure for the file system of the new Linux server, showing how you would set up partitions or separate hard disks and devices to accommodate each of the needs mentioned above. Prepare sample entries for an `fstab` file showing the options that you would likely use for each mounted file system. Include information about how you would configure the swap space on the devices you choose to use.

2. After using the system for several months, you notice that the `df` command shows that one file system is at more than 95% capacity. Describe some steps you might take to remedy this problem. How would your actions vary depending on which of the file systems was at 95%?
3. After running smoothly for about a year, one of the hard disks fails. Fortunately, you have backups of all data. How does the file system arrangement that you have designed assist in getting the system running again?

